Spark で並列分散処理の体験

総合情報基盤センター 教授 布村 紀男

近年，各種方面でビッグデータの話題が良く取り上げられている．Google の MapReduce と Goole File System の論文を契機に Hadoop に代表される巨大なデータ解析に必要なシステムが開発されている．その中でも最近人気急上昇の Apache Spark について，ビッグデータやデータサイエンスに全く精通していない素人の筆者がミニマムクラスタ環境作成し，サンプルプログラムを少し動かしてみた体験を紹介する．

キーワード：Apache Spark, RDD, Python, Scala, RaspberryPi 2

1．はじめに

フリー百科事典 ウィキペディア（Wikipedia）[1]によれば，"並列分散処理（へいれつぶんさんしょり，parallel distributed processing）とは，複数の分散された処理ユニットが同時並行的に情報処理を行うこと．また，そうした情報処理の見方によって人間の認知プロセスの解明を目指す研究アプローチ・・・"とある．HPC(High Performance Computing)分野では，分散メモリ型並列処理として MPI(Message Passing Interface)，共有メモリシステム上でスレッド並列演算として OpenMP はスレッド並列演算を行う業界標準仕様である．最近ではマルチコア CPU/GPU(Graphics Processing Unit)のための OpenCL 並列コンピューティング標準フレームワークが登場している．一方,Web 上などで収集された巨大な分散データの活用として，分散コンピューティング支援目的でGoogleによって2004年に論文発表された MapReduce が有名である．その MapReduce の概念から Hadoop が誕生し，分散ファイルシステム HDFS(Hadoop Distributed File System)，並列分散処理フレームワーク(MapReduce Framework)が提供されている．しかし，Hadoop は使いにくく，処理が非効率といった問題点も指摘され，改善を求められるに至っていた．

2．Apache Spark の特徴

Apache Spark(以降 Spark)は，ビッグデータの活用を支援する大注目のオープンソース並列分散処理基盤であり 高速で汎用的であることを目標に開発されたクラスタシステムである．図-1 に Spark の全体構造[3]を表す．SQL とデータフレームには Spark SQL，リアルタイムストリーム処理における Spark Streaming，機械学習には MLlib，そして，グラフ処理に対しての Graphx，といった豊富なコンポーネントが含まれている．また，Scala, Java, Python そして R の高水準な API が提供されており，柔軟な開発が可能となっている．

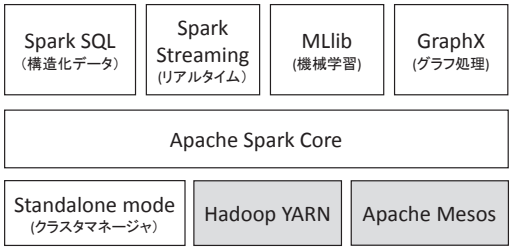

図-1 Apache Spark の全体構造

Spark ではデータを扱う上で核なる耐障害性分散データセット RDD(Resilient Distributed Datasets)の概念は重要である． Spark は RDD のデータを自動的にクラスタ環境で分散させ，並列化処理を行う．クラスタ環境ではクラスタマネジャ(master 機)から worker 群にタスクを割り当て実行する(図-2)．

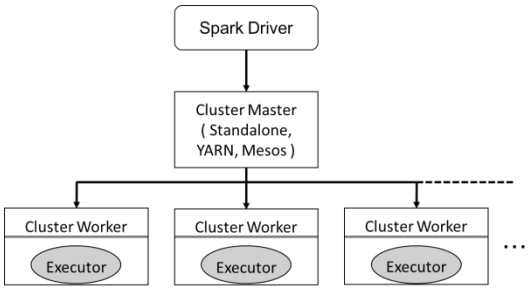

図-2 クラスタでの実行構造

Spark は ad hock な解析にも対応できように，インタラクティブシェルとして Scala と Python 言語の対話シェルが spark-shell と pyspark で用意され

ている.

３．環境構築

今回は，低コストで手軽にミニマムなクラスタ実験環境を作成することを検討した結果，１台 6000 円で入手できる Rasberry Pi 2 [5]でクラスタを組むことにした．ただし，メモリが 1GB で，CPU も ARM Cortex-A7 コア 4 で貧弱なので，Spark の稼動環境としては不安ではあったが，作業を進めた．環境構築について，誰でも考えることが同じなのか，先人の方々[5-7]が Rasberry Pi 2 での構築に関して Web 上で公開していたので，それをお手本にして以下の手順で環境構築を実施した.

３．１ Rasberry Pi の設定

導入 OS は Debian 系 Linux の Raspbian (2015-11-21-raspbian-jessie.zip)を用いた．[2]よりダウンロードした zip ファイルを展開し，イメージファイルを Win32DiskImager 等により，microSD カードに書き込み，OS のインストールが完了する．miscroSD カードを Rabpberry Pi 2 に装着し，OS の起動から無事にデスクトップが表示されたのを確認後，Terminal から初期設定を root 権限(sudo)で raspi-config コマンドにて行う.

```
$ sudo raspi-config
```

(1) ファイル容量の拡張

[1. Expand Filesystem]を選択

今回使用の microSD カードの容量は 8GB であるが、標準では 4GB 弱しか認識していない． OS インストール後は空きがほとんどないので，ファイル容量を拡張する.

(2) タイムゾーンの設定

[5 Internationalisation Options]を選択

[I2 Change Timezone]→[Asia]→[Tokyo]を順に[Enter]ボタンを押下して選択する.

(3) キーボードレイアウトの設定

[I3 Change Keyboard Layout]でキーボード種類の選択をする.

[Generic 105 - key (Intel) PC]→[Other]→[Japanease]→[Japanease-Japanease (OADG109A)]→[-The Default for the Keyboard layout]→[-No Compose key]→[Yes]

(4) 固定 IP アドレスの指定

今回の 5 台のクラスタ構成は，以下のように IP アドレスとホスト名を割り当てた.

1. master 機 192.168.0.10　master
2. worker 機 192.168.0.11　worker1
3. worker 機 192.168.0.12　worker2
4. worker 機 192.168.0.13　worker3
5. worker 機 192.168.0.14　worker4

デフォルトでは DHCP により動的に IP アドレスが割り当てられるが，ここでは静的に IP アドレスに指定するように設定を変更する．まず，master 機の/etc/dhcpd.conf に編集する．続いて worker 群も同様に行う.

```
interface eth0
static ip_address=192.168.0.10/24
static routers=192.168.0.1
static domain_name_servers=192.168.0.1
```

(5) ホスト名の変更

デフォルトのホスト名は，raspberrpi となっているため sudo で/etc/hosts, /etc/hostname を編集する．/etc/hosts に個々のマシンの IP アドレスとホスト名を追加する.

```
$ sudo vi /etc/hosts
```

３．２ Apache Spark のインストール

Spark は Java6 以上および Python2.6 以上であれば動くので，各バージョンを確認する.

```
$ python -V
Python 2.7.9
$ java -version
java version "1.8.0"
Java(TM) SE Runtime Environment (build 1.8.0-b132) Java HotSpot(TM) Client VM (build 25.0-b70, mixed mode)
```

ビルド済みの spark-1.5.2-bin-hadoop2.6.tgz を

```
http://ftp.jaist.ac.jp/pub/apache/spark/spark-1.5.2/spark-1.5.2-bin-hadoop2.6.tgz
```

からダウンロードし，インストールを行う.

専用ユーザ spark を作成することが推奨されているので[5]ユーザ作成し，その後作業はすべて spark ユーザで実施する.

```
$ sudo adduser spark
```

spark ユーザを sudo ユーザに設定する.

```
$ sudo usermod -G sudo spark
```

圧縮ファイルを/home/spark 以下に展開するだけで spark-1.5.2-bin-hadoop2.6 のインストールが終わる.

```
$ su - spark
$ tar xvfz ~pi/spark-1.5.2-bin-hadoop2.6.tgz
$ cd spark-1.5.2-bin-hadoop2.6
```

3．3 クラスタ環境設定

Spark には，クラスタマネージャ(Hadoop YARN, Apache Mesos など)や Amazon EC2 でクラスタを起動するスクリプトも含まれている．さらに Spark のパッケージには Standalone と呼ばれるクラスタマネージャが組み込まれている．今回は簡易的にこれを利用することにした．master 機および worker 機(Slave)間では SSH で通信を行う必要があるため，まず，master 機で SSH の鍵生成を行う．

```
$ ssh-keygen
```

/home/spark/.ssh 以下に秘密鍵の id_rsa と公開鍵の id_rsa.pub が作られる．次に，この公開鍵を各 worker 群に登録する．

```
$ ssh-copy-id spark@worker1
```

worker1 機の/home/spark/.ssh/authorized_key に master 機の公開鍵がコピーされる．

master 機の conf/slaves ファイルに worker 機のホスト名または IP アドレスを記述する．今回は master 機も worker として登録した．

```
$ vi conf/slaves
master
worker1
worker2
worker3
worker4
```

全マシンでクラスタ環境の設定ファイル conf/spark-env.sh の編集を行う．今回は以下のように設定した．

```
$ vi conf/spark-env.sh
SPARK_MASTER_IP=master
SPARK_WORKER_MEMORY=512m
```

クラスタ環境の起動・停止は，master 機で次のスクリプトを実行する．

```
$ sbin/start-all.sh       (起動)
$ sbin/stop-all.sh        (停止)
```

クラスタ環境の稼動状況は，図-3 のように，URL http://master:8080 で見ることがきる．

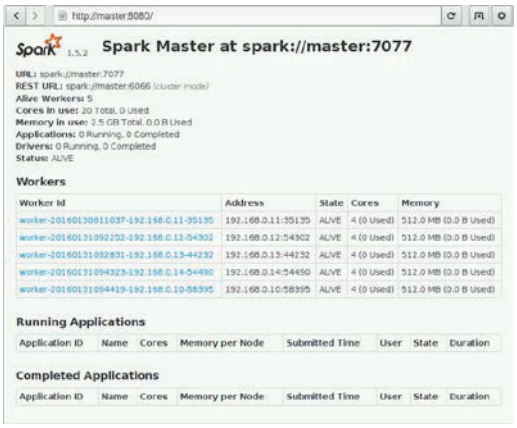

図-3 Apache Spark の起動確認

4．サンプルプログラムよる実行・動作確認

環境構築を終えてから，まずは定番の Scala 言語サンプルであるモンテカルロ法でのπを求めるプログラム SparkPi を単一で走らせて動作確認する．run-example スクリプトを使って実行した．

・Java(Scala)サンプルの実行

```
$ cd spark-1.5.2-bin-hadoop2.6
$ bin/run-example SparkPi 10
```

図-4 サンプルプログラム SpakPi の実行結果

・Spark shell(scala)の実行

```
$ bin/spark-shell
$ sc.textFile("README.md").count
```

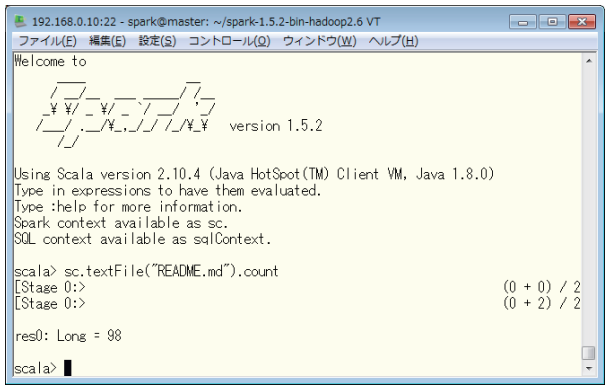

図-5 Scala Shell の起動と実行

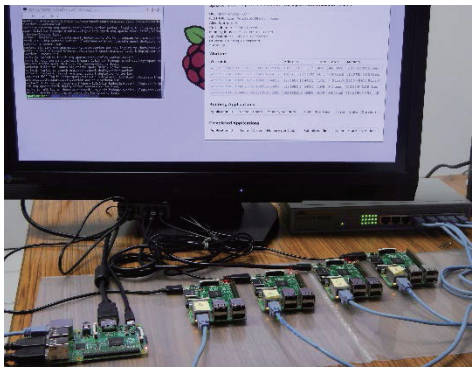

図-6 Raspberry Pi2 クラスタ実験環境

・クラスタ環境での実行

単体マシンおよびクラスタ環境でサンプルプログラム GroupByTest[8]を使って実行し，比較してみた．単体では，--master local を指定する．一方，クラスタ環境では，--master spark://master:7070 を指定する．末尾の引数「50」はテストデータ数を表す．Driver および Executor の割り当てメモリはデフォルトでは，各 1GB であるが，ここでは，256MB を指定した．詳細な設定は，Spark のドキュメントに記述がある[2].

(a) 単体の場合 (worker 数=1)

```
$ bin/spark-submit --driver-memory 256m
--executor-memory 256m --class
org.apache.spark.examples.GroupByTest --
master local
lib/spark-examples-1.5.2-hadoop2.6.0.jar 50
```

(b) クラスタ環境の場合 (worker 数=5)

```
$ bin/spark-submit --driver-memory 256m
--executor-memory 256m --class
org.apache.spark.examples.GroupByTest --
master local
lib/spark-examples-1.5.2-hadoop2.6.0.jar 50
```

(a) の実行結果

```
16/01/31 09:55:33 INFO DAGScheduler: Job 1 finished:
count at GroupByTest.scala:52, took 40.087151 s
50000
```

(b) の実行結果

```
16/01/31 10:02:49 INFO DAGScheduler: Job 1 finished:
count at GroupByTest.scala:52, took 8.765832 s
50000
```

得られた結果から約 4.6 倍速くなっていることがわかる．さらに図-7 に worker 数と 5 回の実行時間の平均値を示す．worker 数が増加してもリニアにはスケールされず，worker 数 4 以上では思ったほどパフォーマンスは期待できなかった．この要因として，通信ネットワークのトラフィックおよびストレージである microSD メモリの個体差や性能差が考えられる．高速ネットワーク，高性能なクラスタサーバ構成が準備できればスケーラビリティは向上するだろう．

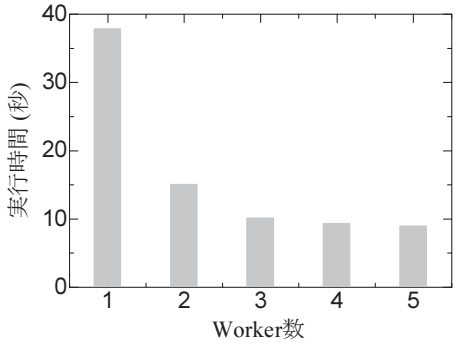

図-7 worker 数と実行時間の関係

5．おわりに

今回は環境構築がメインになり，分散アプリケーションのサンプルプログラムを試しに走らせてみるにとどまった．今後，時間が許せば，分散アプリケーション作成，実行しながら，Spark のアプリ開発およびクラスタ上での利用について考えてみたいと思っている．しかし，その前に Scala 言語を学習することが先かもしれない．本稿で Apache Spark に興味を持っていただければ幸いである．

参考文献

[1] https://ja.wikipedia.org/

[2]Holden Karau,ら著「初めての Spark」オライリー・ジャパン

[3] “Apache Spark ™ Lightning-fast cluster computing” http://spark.apache.org

[4] “Raspberry Pi - Teach, Learn, and Make with Raspberry Pi” https://www.raspberrypi.org/

[5] “Installing Apache Spark on a Raspberry Pi 2” https://darrenjw2.wordpress.com/2015/04/17/installing-apache-spark-on-a-raspberry-pi-2/

[6] “Raspberry Pi 2 で Apache Spark 環境構築” http://make-muda.weblike.jp/2015/05/2767/

[7]“Raspberry Pi2 で Apache Spark を動かしてみた” http://qiita.com/sohatach/items/61c3ff77bba57343c0c6

[8]https://github.com/apache/spark/blob/v1.5.2/examples/src/main/scala/org/apache/spark/examples/GroupByTest.scala